CSE 331 Final Exam Sample Solution 6/10/14

Question 1. (8 points) Suppose we have several Java classes to represent students:

```
class Student extends Object { ... }
class CSEStudent extends Student { ... }
```

Now suppose we have the following variables (which we assume are initialized elsewhere):

```
List<Student> ls;
List<? extends Student> les;
List<? super Student> lss;

List<CSEStudent> lcse;
List<? extends CSEStudent> lecse;
List<? super CSEStudent> lscse;

Student scholar;
CSEStudent hacker;
```

For each of the following, circle OK if the statement is legal or circle ERROR if the Java type checker will indicate an error when it is compiled.

OK (ERROR) `ls = lcse;`

OK (ERROR) `les = lscse;`

OK (ERROR) `lcse = lscse;`

OK (ERROR) `les.add(scholar)`

OK (ERROR) `lscse.add(scholar);`

(OK) ERROR `lss.add(hacker);`

OK (ERROR) `scholar = lscse.get(0);`

(OK) ERROR `hacker = lecse.get(0);`

Question 2. (10 points) hashCodes. Suppose we have the following code for class `Student`, including an `equals` function that considers two students to be "equal" if they have the same graduation year and the same major:

```
public class Student {
  private String name;
  private int graduationYear;
  private String major;
  private List<String> minors;

  // two students are equal if they have the same
  // major and graduation year
  @Override
  public boolean equals(Object o) {
    if (!(o instanceof Student))
      return false;
    Student other = (Student) o;
    return this.graduationYear == other.graduationYear &&
      this.major.equals(other.major);
  }
}
```

(a) (8 points) For each of the following possible `hashCode` methods, circle OK if it is a correct implementation of `hashCode` for this class. Circle ERROR if it is not correct.

OK ERROR
```
public int hashCode() {
  return 31;
}
```

OK ERROR
```
public int hashCode() {
  return major.length() + graduationYear;
}
```

OK ERROR
```
public int hashCode() {
  return major.length() + name.length();
}
```

OK ERROR
```
public int hashCode() {
 return 31*graduationYear + major.length();
}
```

(b) (2 points) Which of the above four methods is likely to yield the best quality `hashCode` for this class? Give a 1- or 2-sentence justification for your answer.

The last one. It is more likely to produce different hash codes for `Student` objects that are not equal.

Question 3. (7 points) Another generic question. One of the interns is trying to implement a generic class to provide an expandable container, somewhat like a list. One of the methods is supposed to expand the underlying array so it has at least as many elements as required. Here is that portion of the code:

```
class ExpandableContainer<E> {

  E[] items;      // instance variable

  // ensure that items has room for at least n
  // elements and expand it if it doesn't
  public void ensureCapacity(int n) {

    if (n <= items.length)

      return;

    // allocate a new array that has n elements

    E[] newItems = new E[n];

    for (int i = 0; i < items.length; i++)

      newItems[i] = items[i];

    items = newItems;

  }

}
```

(a) Error: cannot create an array of generic type. To fix, replace this with: `(E[])new Object[n];`

Unfortunately, this code doesn't compile. (a) (4 points) In the above code, show where the error or errors are located and briefly describe them. Then show how to fix them so the code will compile without errors. (Do not rewrite the method – just show the relatively small fixes needed.)

(b) (3 points) After your repairs have been made to the code, will the code compile without any warning messages? If not, what warnings would the compiler produce and where, and is there any way to eliminate them?

No. The compiler will still produce an "unchecked cast" warning for the `(E[])` cast in the repaired code. A `@SuppressWarnings('unchecked')` annotation can be used to suppress the message, but the underlying cause is still there. The cast cannot be checked at runtime because the generic type information is erased.

Question 4. (10 points) Exceptions and assertions. Here are parts of three possible specifications for a method to calculate square roots:

SA
```
// requires: x >= 0
// returns: approximation to sqrt(x)
```

SB
```
// returns: approximation to sqrt(x)
// throws:  IllegalArgumentException if x < 0
```

SC
```
// returns: approximation to sqrt(x) if x >= 0
//          or Double.NaN (not-a-number) if x < 0
```

And here are two possible implementations of a routine to calculate square roots.

IMP1
```
double sqrt(double x) {
  if (x < 0) throw new IllegalArgumentException();
  return approximation to square root of x;
}
```

IMP2
```
double sqrt(double x) {
  if (x < 0)
     return Double.NaN;
  else
     return approximation to square root of x;
}
```

(a) (6 points) In the table below, put an X in each square where the implementation to the left satisfies the specification given at the top

	SA	SB	SC
IMP1	**X**	**X**	
IMP2	**X**		**X**

(b) (4 points) Of the three specifications given above, which would be most suitable to be included in a general-purpose library intended for wide use, like the Java `Math` class, and why? Give a 1- or 2-sentence justification for your answer. If two or more of the specifications are equally good, give a justification for that answer.

Either SB or SC would be better than SA since they are stronger specifications that specify behavior for all possible inputs rather than allowing unpredictable operation if the client fails to meet the precondition in SA.

[Which one is best depends on how the library is to be used and whether an exceptional value (NaN) is preferred to producing an exception if the input is negative. The Java `Math.sqrt` library routine uses SC.]

Question 5. (8 points) The Java Collection classes allow users to wrap a collection so it can be viewed but not modified. Suppose we have a list `lst` that has type `List<`*Foo*`>` for some unknown type *Foo*. We then execute the statement

```
List<Foo> unlst = Collections.unmodifiableList(lst);
```

The new variable `unlst` can be used to view the contents of the original list, but if any method is called to attempt to change it (for example `unlst.add(x)` or `unlst.remove(y)`) an `UnsupportedOperationException` is thrown.

(a) (4 points) Clearly, both `lst` and `unlst` have the same Java type `List<`*Foo*`>`. But do these lists have exactly the same true specification type? Is the type of one of these lists a true subtype of the other? Or are the actual types of the lists incomparable (i.e., neither is a true subtype of the other)? Give a brief explanation.

The `unmodifiableList` does not have the same true specification as the original `List`. They have different specifications for mutator methods like `add()` and `remove()`. Neither specification is a stronger or weaker version of the other – they are not comparable (i.e., invariant).

(b) (4 points) Suppose that the original variable `lst` was a private instance variable in a class, and the class includes an observer method that returns the unmodifiable list `unlst` created as in part (a) above. Are there any potential representation exposure problems with this method?

Yes, there is a representation exposure problem if the items stored in the list are mutable. In that case client code could modify individual list elements even though it cannot modify the list itself.

Question 6. (12 points) Swing! Here is most of the code for a small application that displays two buttons in a window. You need to complete some code to get the application to print the correct messages when the buttons are clicked.

```
public class Buttons {
  public static void main (String[] args) {
    JFrame frame = new JFrame("Finals");
    frame.setDefaultCloseOperation(
                    WindowConstants.EXIT_ON_CLOSE);

    JButton button1 = new JButton("button1");
    button1.setActionCommand("uno");
    JButton button2 = new JButton("button2");
    button2.setActionCommand("duo");

    ButtonListener buttonListener = new ButtonListener();
    // make buttonListener the listener object for button1 and button2

    button1.addActionListener(buttonListener);

    button2.addActionListener(buttonListener);

    frame.add(button1, BorderLayout.NORTH);
    frame.add(button2, BorderLayout.SOUTH);
    frame.pack();
    frame.setVisible(true);
  }
}
```

(Class `ButtonListener` will be implemented on the next page.)

(a) (5 points) Write code in the blank box above so that when either button is clicked the object named `buttonListener` will be notified that a click has occurred. (This does not require much code at all – we just left a lot of space in case you need it.)

Question 6. (cont.) (b) (7 points) Fill in the body of class `ButtonListener` below so whenever `button1` is clicked the message "button1 rocks!" is printed on `System.out` and whenever `button2` is clicked the message "button2 rules!" is printed on `System.out`.

```
class ButtonListener implements ActionListener {
  // write your code below

  public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equals("uno")) {
      System.out.println("button1 rocks!");
    } else if (e.getActionCommand().equals("duo")) {
      System.out.println("button2 rules!");
    }
  }
}
```

A bit of reference information:

The Java library interface `ActionListener` includes just this one method:

```
void actionPerformed(ActionEvent e)
```

Class `ActionEvent` contains the following methods:

```
String getActionCommand()
int    getModifiers()
long   getWhen()
String paramString()
```

Question 7. (10 points) Suppose we are building a program to play Chess. Being conscientious CSE 331 graduates, we're using the Model-View-Controller (MVC) pattern to structure our code.

For each of the following operations that are part of the program, indicate which part of the MVC architecture would implement that operation by circling M, V, and/or C. If more than one part of the architecture would be responsible for that part of the program circle all of the parts. If none of the parts of the architecture apply, leave the answer blank.

M (V) (C) Move a piece across the screen as the player drags it around with the mouse.

(M) V C Determine if a proposed move made by the player is legal.

M (V) (C) Provide a chat box so the player can talk to his/her opponent.

(M) V C Compute the next move to be made when the computer is one of the players.

M (V) C Draw the game board on the screen.

Question 8. (4 points) What is the key difference between testing and debugging? Or are they really the same thing? (Be brief, please)

Testing is the process of demonstrating the existence of defects or providing confidence that they do not appear to be present.

Debugging is the process of discovering the cause of a defect and fixing it.

Question 9. (4 points) When fixing a bug, a key step is to create a small test case that demonstrates the problem. Conventional wisdom says that you should then add that test to the permanent test suite to use as a regression test in the future. Why? Doesn't this just add useless bulk to the collection of tests? (A couple of sentences should be enough to answer.)

If a bug occurs it is because of some error or misunderstanding. As the code is modified in the future, there's a decent chance that the same error might be introduced again for the same reasons, or because the person working on the code is not aware of or has forgotten the past history. Adding the test to the permanent collection increases confidence that future changes do not re-introduced the old problem.

Further, if there is a bug and no test caught the problem previously then some part of the input domain was not properly covered by a test. Retaining the test will provide better coverage for that part of the input domain.

Question 10. (10 points) Testing. Suppose we have the following method in one of our classes:

```
// Given non-negative integer n, return n!
int factorial(int n) {
  int num = n;
  int fac = 1;
  while (num != 1) {
    fac *= num;
    num--;
  }
  return fac;
}
```

(b) To fix the bug, change 1 to 0 here.

For some reason this method goes into an infinite loop every now and then and it is necessary to kill the program to get it to stop.

(a) (7 points) Produce a simple, small test case to demonstrate the failure. Complete the following jUnit test so it will reproduce the problem.

```
@Test(timeout=1000)
public void testFactorialNonTermination() {

   int result = factorial(0);

   assertEquals("factorial(0)", 1, result);

}
```

(b) (3 points) What is the cause of the problem? Indicate the problem by writing on the code above and show how it to fix it.

Question 11. (5 points) System design and implementation. The two main strategies for implementing a complex system are top-down and bottom-up. These two strategies have different strengths. For each of the statements below, circle top-down or bottom-up to indicate which implementation strategy is the best match. If both strategies are equally good, circle both. You do not need to justify your answers.

Better at showing visible progress (i.e, demos and prototypes) to customers and implementers.

(top-down) bottom-up

Likely to require more non-deliverable code (tests, stubs, drivers, mock objects)

(top-down) bottom-up

Better at revealing possible resource problems or performance bottlenecks early in the project.

top-down (bottom-up)

Better at revealing problems with basic design decisions early in the project.

(top-down) bottom-up

Potentially can require more effort to integrate each new component as it becomes available.

top-down (bottom-up)

Question 12. (4 points) IDEs like Eclipse have built-in tools to automate program compilation and the other steps needed to build a program. Why then do most projects also use an external build tool like ant or make? Why not just let the IDE take care of the job and handle the work needed to build things? (Be brief – a sentence or two is enough to get the main ideas across.)

It is important that everyone on a project use a reproducible build process. Otherwise there can be problems if different IDEs, or even different versions of the same IDE, build the project in slightly different ways, leading to subtle differences and hard-to-diagnose bugs.

Question 13. (8 points) Design patterns – the traditional last question. Here is an alphabetical list of some design patterns we studied. Note that some patterns are more specific instances of others.

Adapter, Builder, Composite, Decorator, Factory, Flyweight, Iterator, Intern, Interpreter, Model-View-Controller (MVC), Observer, Procedural, Prototype, Proxy, Singleton, Visitor, Wrapper

For each of the situations below, list all of the design pattern or patterns from the list that are used in that situation. (There is at least one pattern that is appropriate for each.)

(a) Add a scroll bar to an existing window object in Swing.

Decorator

(b) We have an existing object that controls a communications channel. We would like to provide the same interface to clients but transmit and receive encrypted data over the existing channel.

Proxy

(c) We have a computer simulation of a bicycle involving wheels, spokes, and so forth, and we want to be able to treat all of these parts and subassemblies of the bicycle uniformly as bicycle components in some parts of the code.

Composite

(d) When the user clicks the "find path" button in the Campus Maps application (hw9), the path appears on the screen.

MVC and Observer

In retrospect it was probably a mistake to say "list all patterns that apply to this situation" since many answers we read included the right answer along with a bunch of patterns that were at best tangentially related to the situation but usually were not relevant at all. We did give partial credit in those situations when it seemed appropriate.

Have a great summer! See you in the fall!!